Theory and Applications of Mathematics & Computer Science

(ISSN 2067-2764, EISSN 2247-6202)
http://www.uav.ro/applications/se/journal/index.php/tamcs

Theory and Applications of Mathematics & Computer Science 6 (2) (2016) 170 – 186

Modeling Evolution by Evolutionary Machines: A New Perspective on Computational Theory and Practice

Mark Burgin[a,*], Eugene Eberbach[b]

[a]*Dept. of Mathematics, University of California, 405 Hilgard Avenue, Los Angeles, CA 90095, USA*
[b]*Dept. of Computer Science, Robotics Engineering Program, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609, USA*

Abstract

The main goal of this paper is the further development of the foundations of evolutionary computations, connecting classical ideas in the theory of algorithms and the contemporary state of art in evolutionary computations. To achieve this goal, we develop a general approach to evolutionary processes in the computational context, building mathematical models of computational systems, called evolutionary machines or automata. We introduce two classes of evolutionary automata: basic evolutionary automata and general evolutionary automata. Relations between computing power of these classes are explored. Additionally, several other classes of evolutionary machines are investigated, such as bounded, periodic and recursively generated evolutionary machines. Different properties of these evolutionary machines are obtained.

Keywords: Turing unorganized machines, evolutionary computation, evolutionary automata, evolutionary Turing machines, evolutionary finite automata, evolutionary inductive Turing machines.
2010 MSC: 03D10, 03D28.

1. Introduction

The classical theory of algorithms has been developed under the influence of Alan Turing, who was one of the founders of theoretical computer science and whose model of computation, which is now called Turing machine, is the most popular in computer science. He also had many other ideas. In this report the National Physical Laboratory in 1948 (Turing, 1992), Turing proposed a new model of computation, which he called unorganized machines (u-machines). There were two types of u-machines: based on Boolean networks and based on finite state machines.

- A-type and B-type u-machines were Boolean networks made up of a fixed number of two-input NAND gates (neurons) and synchronized by a global clock. While in A-type u-machines the connections between neurons were fixed, B-type u-machines had modifiable

*Corresponding author
Email addresses: mburgin@math.ucla.edu (Mark Burgin), eeberbach@wpi.edu (Eugene Eberbach)

switch type interconnections. Starting from the initial random configuration and applying a kind of genetic algorithm, B-type u-machines were supposed to learn which of their connections should be on and which off.

- P-type u-machines were tapeless Turing machines reduced to their finite state machine control, with an incomplete transition table, and two input lines for interaction: the pleasure and the pain signals.

Although Turing never formally defined a genetic algorithm or evolutionary computation, in his B-type u-machines, he predicted two areas at the same time: neural networks and evolutionary computation (more precisely, evolutionary artificial neural networks), while his P-type u-machines represent reinforcement learning. However, this work had no impact on these fields (Eberbach *et al.*, 2004), although these ideas are one of the (almost forgotten) roots of evolutionary computation.

Evolutionary computation theory is still very young and incomplete. Until recently, evolutionary computation did not have a theoretical model that represented practice in this domain. Even though there are many results on the theory of evolutionary algorithms (see, e.g., (Michalewicz & Fogel, 2004), (He & Yao, 2004), (Holland, 1975), (Rudolph, 1994), (Wolpert & Macready, 1997), (Koza, 1992, 1994; Koza *et al.*, 1999), (Michalewicz, 1996), very little has been known about expressiveness, or computational power, of evolutionary computation (EC) and its scalability. Of course, there are many results on the theory of evolutionary algorithms (again see, for instance, (Michalewicz & Fogel, 2004), (He & Yao, 2004), (Holland, 1975), (Rudolph, 1994), (Wolpert & Macready, 1997), (Koza, 1992, 1994; Koza *et al.*, 1999), (Michalewicz, 1996)). Studied in EC theoretical topics include convergence in the limit (elitist selection, Michalewiczs contractive mapping GAs, (1+1)-ES), convergence rate (Rechenbergs 1/5 rule), the Building Block analysis (Schema Theorems for GA and GP), best variation operators (No Free Lunch Theorem). However, these authors do not introduce automata models - rather they apply a high-quality mathematical apparatus to existing process models, such as Markov chains, etc. They also cover only some aspects of evolutionary computation like convergence or convergence rate, neglecting for example EC expressiveness, self-adaptation, or scalability. In other words, EC is not treated as a distinct and complete area with its own distinct model situated in the context of general computational models. This means that in spite of intensive usage of mathematical techniques, EC lacks more complete theoretical foundations. As a result, many properties of evolutionary processes could not be precisely studied or even found by researchers. Our research is aimed at filling this gap to define more precisely conditions under which evolutionary algorithms will work and will be superior compared to other optimization methods.

In 2005, the evolutionary Turing machine model was proposed to provide more rigorous foundations for EC (Eberbach, 2005). An evolutionary Turing machine is an extension of the conventional Turing machine, which goes beyond the Turing machine and belongs to the class of super-recursive algorithms (Burgin, 2005). In several papers, the authors studied and extended the ETM (evolutionary Turing machine) model to reflect cooperation and competition (Burgin & Eberbach, 2008), universality (Burgin & Eberbach, 2009*b*), self-evolution (Eberbach & Burgin, 2007), and expressiveness of evolutionary finite automata (Burgin & Eberbach, 2009*a*), (Burgin & Eberbach, 2012).

In this paper, we continue developing a general approach to evolutionary processes in the computational context constructing mathematical models of the systems, functioning of which is based on evolutionary processes, and study properties of such systems with the emphasis on their generative power. Two classes are introduced in Section 2: basic evolutionary automata and general evolutionary automata. Relations between computing power of these classes are explored in Section 3. Additionally, several other classes of evolutionary machines are investigated, such as bounded, periodic and recursively generated evolutionary machines. Different properties of these evolutionary machines are obtained. Section 4 contains conclusions and some open problems.

2. Modeling Evolution by Evolutionary Machines

Evolutionary algorithms describe artificial intelligence processes based on the theory of natural selection and evolution. Evolutionary computation is directed by evolutionary algorithms. In technical terms, an evolutionary algorithm is a probabilistic beam hill climbing search algorithm directed by the chosen fitness function. It means that the beam (population size) maintains multiple search points, hill climbing implies that only a current search point from the search tree is remembered and used for optimization (going to the top of the hill), and the termination condition very often is set to the optimum of the fitness function.

Let X be the representation space, also called the optimization space, for species (systems) used in the process of optimization and a fitness function $f : X \to \mathbb{R}^+$ is chosen, where $\mathbb{R}^+$ is the set of nonnegative real numbers.

Definition 2.1. *A generic evolutionary algorithm EA* can be represented as the collection $EA = (X, s, v, f, R, X[0], F)$ and described in the form of the functional equation (recurrence relation) R working in a simple iterative loop in discrete time t, defining generations $X[t]$, $(t = 0, 1, 2, 3, ...)$ with $X[t + 1] = s(v(X[t]))$, where

- $X[t] \subseteq X$ is a population under a representation consisting of one or more individuals from the set X (e.g., fixed binary strings for genetic algorithms (GAs), finite state machines for evolutionary programming (EP), parse trees for genetic programming (GP), vectors of reals for evolution strategies (ES)),
- s is a selection operator (e.g., truncation, proportional, tournament),
- v is a variation operator (e.g., variants and compositions of mutation and crossover),
- $X[0]$ is an initial population,
- $F \subseteq X$ is the set of final populations satisfying the termination condition (goal of evolution). The desirable termination condition is the optimum in X of the fitness function $f(x)$, which is extended to the fitness function $f(X[t])$ of the best individual in the population $X[t] \subseteq F$, where $f(x)$ typically takes values in the domain of nonnegative real numbers. In many cases, it is impossible to achieve or verify this optimum. Thus, another stopping criterion is used (e.g., the maximum number of generations, the lack of progress through several generations.).

The above definition is applicable to all typical EAs, including GA, EP, ES, GP. It is possible to use it to describe other emerging subareas like ant colony optimization, or particle swarm optimization. Of course, it is possible to think and implement more complex variants of evolutionary algorithms.

Evolutionary algorithms evolve population of solutions x, but they may be the subject of self-adaptation (like in ES) as well. For sure, evolution in nature is not static, the rate of evolution fluctuates, their variation operators are subject to slow or fast changes, and its goal (if it exists at all) can be a subject of modifications as well.

Formally, an evolutionary algorithm looking for the optimum of the fitness function violates some classical requirements of recursive algorithms. If its termination condition is set to the optimum of the fitness function, it may not terminate after a finite number of steps. To fit it to the conventional algorithmic approach, an artificial (or somebody can call it pragmatic) stop criterion has had to be added (see e.g., (Michalewicz, 1996), (Michalewicz & Fogel, 2004), (Koza, 1992, 1994; Koza *et al.*, 1999)). To remain recursive, i.e., to give some result after a finite number of steps, the evolutionary algorithm has to reach the set F of final populations satisfying the termination condition after a finite number of generations or to halt when no visible progress is observable. Usually this is a too restrictive condition, and naturally, in a general case, evolutionary algorithms form a special class of super-recursive algorithms.

To formalize the concept of an evolutionary algorithm in mathematically rigorous terms, we define a formal algorithmic model of evolutionary computation - an evolutionary automaton also called an evolutionary machine.

Let K be a class of automata working with words in an alphabet E. It means that the representation or optimization space X is the set E^* of all words in an alphabet E.

Definition 2.2. *A basic evolutionary K-machine* (BEM), also called *basic evolutionary K-automaton*, is a (possibly infinite) sequence $E = \{A[t]; t = 0, 1, 2, 3, ...\}$ of automata $A[t]$ from K each working on the population $X[t] \subseteq X(t = 0, 1, 2, 3, ...)$ where:

- the automaton $A[t]$ called a component, or more exactly, a level automaton, of E represents (encodes) a one-level evolutionary algorithm that works with the generation $X[t]$ of the population by applying the *variation operators* v and *selection operator* s;
- the zero generation $X[0]$ is given as input to E and is processed by the automaton $A[0]$, so that either $X[0]$ is the result of the whole computation by E when it satisfies the search condition or $A[0]$ generates/produces the first generation $X[1]$ as its output, which goes to the automaton $A[1]$;
- for all $t = 1, 2, 3, ...$, the generation $X[t + 1]$ is obtained by applying the variation operator v and selection operator s to the generation $X[t]$ and these operations are performed by the automaton $A[t]$, which receives $X[t]$ as its input; the generation $X[t+1]$ either is the result of the whole computation by E when it satisfies the search condition or it goes to the automaton $A[t + 1]$;
- the goal of the BEM E is to build a population Z satisfying the search condition.

The desirable search condition is the optimum of the fitness performance measure $f(x[t])$ of the best individual from the population $X[t]$. There are different modes of the EM functioning and different termination strategies. When the search condition is satisfied, then working in the recursive mode, the EM E halts (t stops to be incremented), otherwise a new input population $X[t+1]$ is generated by $A[t]$. In the inductive mode, it is not necessary to halt to give the result (cf. (Burgin, 2005)). When the search condition is satisfied and E is working in the inductive mode, the EM E stabilizes (the population $X[t]$ stops changing), otherwise a new input population $X[t+1]$ is generated by $A[t]$.

We denote the class of all basic evolutionary machines with level automata from K by BEAK.

Definition 2.3. *A general evolutionary K-machine* (GEM), also called *general evolutionary K-automaton*, is a (possibly infinite) sequence $E = \{A[t]; t = 0, 1, 2, 3, ...\}$ of automata $A[t]$ from K each working on generations $X[i] \subseteq X$ where:

- the automaton $A[t]$ called a component, or more exactly, a level automaton, of E represents (encodes) a one-level evolutionary algorithm that works with generations $X[i]$ of the population by applying the variation operators v and selection operator s;
- the zero generation $X[0] \subseteq X$ is given as input to E and is processed by the automaton $A[0]$, which generates/produces the first generation $X[1]$ as its output, which either is the result of the whole computation by E when it satisfies the search condition or it goes to the automaton $A[1]$;
- for all $t = 1, 2, 3, ...$, the automaton $A[t]$, which receives $X[i]$ as its input either from $A[t+1]$ or from $A[t-1]$, then $A[t]$ applies the variation operator v and selection operator s to the generation $X[t]$, producing the generation $X[t+1]$ as its output, which either is the result of the whole computation by E when it satisfies the search condition or it goes either to $A[t+1]$ or to $A[t-1]$. To perform such a transmission, the automaton A[t] uses one of the two techniques: transmission by the output and transmission by the state. In transmission by the output, the automaton $A[t]$ uses two more symbols u_{up} and u_{dw} in its output alphabet, giving one of these symbols as a part of its output in addition to the regular output $X[t+1]$. If this part of the output is u_{up}, then $A[t]$ sends the output generation $X[t+1]$ to $A[t+1]$. If the additional part of the output is u_{dw}, then $A[t]$ sends the output generation $X[t+1]$ to $A[t-1]$. In transmission by the state, the automaton $A[t]$ uses two more symbols u_{up} and u_{dw} as its final-transmission states. In these states the automaton $A[t]$ stops computing and performs the necessary transmission of the output - to the automaton $A[t+1]$ when the state is u_{up} and to the automaton $A[t-1]$ when the state is u_{dw}.
- the goal of the GEM E is to build a population Z satisfying the search condition.

We denote the class of all general evolutionary K-machines GEAK. As any basic evolutionary K-machine is also a general evolutionary K-machine, we have inclusion of classes $BEAK \subseteq GEAK$.

Let us consider some examples of evolutionary K-machines. An important class of evolutionary machines are evolutionary finite automata (Burgin & Eberbach, 2009*a*), (Burgin & Eberbach, 2012). Here K consists of finite automata.

Definition 2.4. *A basic (general) evolutionary finite automaton* (EFA) is a basic (general) evolutionary machine E in which all automata $A[t]$ are finite automata $G[t]$ each working on the population $X[t]$ in generations $t = 0, 1, 2, 3, ...$

We denote the class of all general evolutionary finite automata by GEFA. It is possible to take as K deterministic finite automata, which form the class DFA, or nondeterministic finite automata, which form the class NFA. This gives us four classes of evolutionary finite automata: BEDFA (GEDFA) of all deterministic basic (general) evolutionary finite automata and BENFA (GENFA) of all nondeterministic basic (general) evolutionary finite automata.

Evolutionary Turing machines (Burgin & Eberbach, 2008), (Eberbach, 2005) are form another important class of evolutionary machines.

Definition 2.5. *A basic (general) evolutionary Turing machine* (ETM) $E = \{T[t]; t = 0, 1, 2, 3, ...\}$ is a basic (general) evolutionary machine E in which all automata $A[t]$ are Turing machines $T[t]$ each working on population $X[t]$ in generations $t = 0, 1, 2, 3, ...$

Turing machines $T[t]$ as components of E perform multiple computations (Burgin, 1983). Variation and selection operators are recursive to allow performing level computation on Turing machines.

Definition 2.6. *A basic (general) evolutionary inductive Turing machine* (EITM) $EI = \{M[t]; t = 0, 1, 2, ...\}$ is a basic (general) evolutionary machine E in which all automata $A[t]$ are inductive Turing machines $M[t]$ (Burgin, 2005) each working on the population $X[t]$ in generations $t = 0, 1, 2, ...$

Simple inductive Turing machines are abstract automata (models of algorithms) closest to Turing machines. The difference between them is that a Turing machine always gives the final result after a finite number of steps and after this it stops or, at least, informs when the result is obtained. Inductive Turing machines also give the final result after a finite number of steps, but in contrast to Turing machines, inductive Turing machines do not always stop the process of computation or inform when the final result is obtained. In some cases, they do this, while in other cases they continue their computation and give the final result. Namely, when the content of the output tape of a simple inductive Turing machine forever stops changing, it is the final result.

Definition 2.7. *A basic (general) evolutionary inductive Turing machine* (EITM) $EI = \{M[t]; t = 0, 1, 2, ...\}$ has *order n* if all inductive Turing machines $M[t]$ have order less than or equal to n and at least, one inductive Turing machine $M[t]$ has order n.

We remind that inductive Turing machines with recursive memory are called inductive Turing machines of the first order (Burgin, 2005). The memory E is called n-inductive if its structure is constructed by an inductive Turing machine of the order n. Inductive Turing machines with n-inductive memory are called inductive Turing machines of the order $n + 1$. We denote the class of all evolutionary inductive Turing machines of the order n by $EITM_n$.

Definition 2.8. *A basic (general) evolutionary limit Turing machine* (ELTM) $EI = \{LTM[t]; t = 0, 1, 2, ...\}$ is a basic (general) evolutionary machine E in which all automata $A[t]$ are limit Turing machines $LTM[t]$ (cf. (Burgin, 2005)) each working on the population $X[t]$ in generations $t = 0, 1, 2, ...$

When the search condition is satisfied, then the ELTM EI stabilizes (the population $X[t]$ stops changing), otherwise a new input population $X[t+1]$ is generated by LT $M[t]$. We denote the class of all evolutionary limit Turing machines of the first order by ELTM.

Basic and general evolutionary K-machines from BEAK and GEAK are called unrestricted because sequences of the level automata $A[t]$ and the mode of the evolutionary machines functioning are arbitrary. For instance, there are unrestricted evolutionary Turing machines when K is equal to T and unrestricted evolutionary finite automata when K is equal to FA. However it is possible to consider only basic (general) evolutionary K-machines from BEAK (GEAK) in which sequences of the level automata have some definite type Q. Such machines are called Q-formed basic (general) evolutionary K-machines and their class is denoted by $BEAK^Q$ for basic machines and $GEAKQ$ for general machines. When the type Q contains all finite sequences, we have bounded basic (general) evolutionary K-machines. Some classes of bounded basic evolutionary K-machines are studied in (Burgin & Eberbach, 2010) for such classes K as finite automata, pushdown automata, Turing machines, or inductive Turing machines, i.e., such classes as bounded basic evolutionary Turing machines or bounded basic evolutionary finite automata. When the type Q contains all periodic sequences, we have periodic basic (general) evolutionary K-machines. Some classes of periodic basic evolutionary K-machines are studied in (Burgin & Eberbach, 2010) for such classes K as finite automata, push down automata, Turing machines, inductive Turing machines and limit Turing machines. Note that while in a general case, evolutionary automata cannot be codified by finite words, periodic evolutionary automata can be codified by finite words.

Another condition on evolutionary machines determines their mode of functioning or computation. Here we consider the following modes of functioning/computation.

1. The finite-state mode: any computation is going by state transition where states belong to a fixed finite set.
2. The bounded mode: the number of generations produced in all computations is bounded by the same number.
3. The terminal or finite mode: the number of generations produced in any computation is finite.
4. The recursive mode: in the process of computation, it is possible to reverse the direction of computation, i.e., it is possible to go from higher levels to lower levels of the automaton, and the result is defined after finite number of steps.
5. The inductive mode: the computation goes in one direction, i.e., without reversions, and if for some t, the generation $X[t]$ stops changing, i.e., $X[t] = X[q]$ for all $q > t$, then $X[t]$ is the result of computation.
6. The inductive mode with recursion: recursion (reversion) is permissible and if for some t, the generation $X[t]$ stops changing, i.e., $X[t] = X[q]$ for all $q > t$, then $X[t]$ is the result of computation.
7. The limit mode: the computation goes in one direction and the result of computation is the limit of the generations $X[t]$.
8. The limit mode with recursion: recursion (reversion) is permissible and the result of computation is the limit of the generations $X[t]$.

These modes are complementary to the three traditional modes of computing automata: computation, acceptation and decision/selection (Burgin, 2010). Existence of different modes of computation shows that the same algorithmic structure of an evolutionary automaton/machine E provides for different types of evolutionary computations. We see that only general evolutionary machines allow recursion. In basic evolutionary machines, the process of evolution (computation) goes strictly in one direction. Thus, general evolutionary machines have more possibilities than basic evolutionary machines and it is interesting to relations between these types of evolutionary machines. This is done in the next section. Note that utilization of recursive steps in evolutionary machines provides means for modeling reversible evolution, as well as evolution that includes periods of decline and regression.

3. Computing and Accepting Power of Evolutionary Machines

As we know from the theory of automata and computation, it is proved that different automata or different classes of automata are equivalent. However there are different kinds of equivalence. Here we consider two of them: functional equivalence and linguistic equivalence.

Definition 3.1. (Burgin, 2010)

a. Two automata A and B are functionally equivalent if given the same input, they give the same output.

b. Two classes of automata A and B are functionally equivalent if for any automaton from A, there is a functionally equivalent automaton from B and vice versa.

For instance, it is proved that deterministic and nondeterministic Turing machines are functionally equivalent (cf., for example, (Hopcroft *et al.*, 2001)). Similar results are true for evolutionary automata.

Theorem 3.1. *(Burgin & Eberbach, 2010) For any basic n-level evolutionary finite automaton E, there is a finite automaton AE functionally equivalent to E.*

Here we study relations between basic and general evolutionary machines, assuming that all these machines work in the terminal mode.

Let $P : X \times U \to N$ be a function such that

$$U = \{u_{1,up}, u_{1,dw}, u_{2,up}, u_{2,dw}, ..., u_{k,up}, u_{k,dw}, ...\},$$

$$P(x, u_{k,up}) = k + 1$$

and

$$P(x, u_{k,dw}) = k - 1$$

for any x from optimization space $X = E^*$.

Definition 3.2. (Burgin, 2010) The P-conjunctive parallel composition $\wedge_P A_i$ of the algorithms/automata A_i $(i = 1, 2, 3, ..., n)$ is an algorithm/automaton D such that the result of application of D to any input u is equal to $A_i(u)$ when $P(u) = i$.

This concept allows us to show in a general case of the terminal mode that basic and general evolutionary machines are equivalent.

Theorem 3.2. *If a class K is closed with respect to P-conjunctive parallel composition, then for any general evolutionary K-machine, there is a functionally equivalent basic evolutionary K-machine.*

Proof. Let us consider an arbitrary general evolutionary K-machine $E = \{A[t]; t = 0, 1, 2, 3, ...\}$. We correspond the evolutionary system $H = \{C[t]; t = 0, 1, 2, 3, ...\}$ to the K-machine E. Each component $C[t]$ in H is a system that consists of the automata $C_0[t], C_1[t], C_2[t], C_3[t], ..., C_t[t]$ such that for all $k = 0, 1, 2, 3, ..., t$, the automaton $C_k[t]$ is a copy of the automaton $A[k]$ and it uses the elements $u_{k,up}$ and $u_{k,dw}$ instead of the elements u_{up} and u_{dw} employed by $A[k]$.

The system H has the same search condition as the evolutionary K-machine E and functions in the following way. The zero generation $X[0] \subseteq X$ is given as input to the automaton $C_0[0]$, which is a copy of the automaton $A[0]$ and is processed by the automaton $C_0[0]$, which generates/produces the first generation $X[1]$ as its output. Then $X[1]$ either is the result of the whole computation by H when it satisfies the search condition or it goes to the automaton $C_1[1]$ as its input. In the general case, for all $t = 1, 2, 3, ...$ and $k = 1, 2, 3, ..., t$, the automaton $C_k[t]$ receives $X[t]$ as its input either from $C_{k+1}[t-1]$ when the automaton $A[k]$ receives its input from $A[k+1]$ or from $C_{k-1}[t-1]$ when the automaton $A[k]$ receives its input from $A[k-1]$. Then $C_k[t]$ applies the variation operator ν and selection operator s to the generation $X[t]$ and producing the generation $X[t+1]$. Then either this generation is the result of the whole computation by H when it satisfies the search condition or $C_k[t]$ sends this generation either to $C_{k+1}[t+1]$ when the automaton $A[k]$ sends its output to $A[k+1]$ or to $C_{k-1}[t+1]$ when the automaton $A[k]$ sends its output to $A[k-1]$.

In such a way, the system H simulates functioning of the general evolutionary K-machine $E = \{A[t]; t = 0, 1, 2, 3, ...\}$. Let us prove this by induction on the number of steps that the K-machine E is making.

The base of induction:

Making the first step, the K-machine E receives is the zero generation $X[0] \subseteq X$ as its input, processes it by the first automaton $A[0]$ producing the first generation $X[1]$, which either is the result of the whole computation by E when it satisfies the search condition or it goes to the automaton $A[1]$.

Making the first step, the system H receives the zero generation $X[0] \subseteq X$ as its input, processes it by the first automaton $C_0[0]$ producing the first generation $Z[1]$, which either is the result of the whole computation by H when it satisfies the search condition or it goes to the automaton $C_1[1]$. Because the system H has the same search condition as the evolutionary K-machine E, $C_0[0]$ is a copy of the automaton $A[0]$, while $C_1[1]$ is a copy of the automaton $A[1]$, we have the equality $Z[1] = X[1]$ and the first step of the system H exactly simulates the first step of the K-machine E.

The general step of induction:

We suppose that making $n-1$ steps the system H exactly simulates $n-1$ steps of the K-machine E. It means that making $n-1$ steps, both systems E and H produce the same n-th generation $X[n]$ using automata $A[r]$ $(r \leqslant n-1)$ and $C_r[n-1]$, correspondingly, and this output either is the result of the whole computation by E and by H when it satisfies the search condition or it goes either to

the automaton $A[r+1]$ or to the automaton $A[r-1]$ in E and either to the automaton $C_{r+1}[n]$ or to the automaton $C_{r-1}[n]$ in H.

Then the automaton $A[r+1]$ (or $A[r-1]$) in E produces the next generation $X[n+1]$, applying the variation operator ν and selection operator s to the generation $X[n]$ and producing the next generation $X[n+1]$. When the automaton $A[r+1]$ in E produces the next generation $X[n+1]$, then either this generation is the result of the whole computation by E when it satisfies the search condition or $A[r+1]$ sends this generation either to $A[r+2]$ or to $A[r]$. When the automaton $A[r-1]$ in E produces the next generation $X[n+1]$, then either this generation is the result of the whole computation by E when it satisfies the search condition or $A[r-1]$ sends this generation either to $A[r-2]$ or to $A[r]$.

At the same time, the automaton $C_{r+1}[n]$ (or $C_{r-1}[n]$) applies the variation operator ν and selection operator s to the generation $X[n]$ and producing the generation $Z[n+1]$. When the automaton $C_{r+1}[n]$ in H produces the next generation $Z[n+1]$, then either this generation is the result of the whole computation by H when it satisfies the search condition or $C_{r+1}[n]$ sends this generation either to $C_{r+2}[n+1]$ or to $C_r[n+1]$. When the automaton $C_{r-1}[n]$ in H produces the next generation $Z[n+1]$, then either this generation is the result of the whole computation by H when it satisfies the search condition or $C_{r-1}[n]$ sends this generation either to $C_{r-2}[n]$ or to $C_r[n]$.

Because system H has the same search condition as the evolutionary K-machine E, $C_{r+1}[n]$ is a copy of the automaton $A[r+1]$, while $C_{r-1}[n]$ is a copy of the automaton $A[r-1]$, we have the equality $Z[n+1] = X[n+1]$ and the n-th step of the system H exactly simulates the n-th step of the K-machine E.

Now it is possible to conclude that the system H exactly simulates functioning of the K-machine E. However, the system H is not an evolutionary K-machine. So we need to build a basic evolutionary K-machine B equivalent to H. We can do this using P -conjunctive parallel composition. This composition allows us for all $t = 0, 1, 2, 3, ...$, to substitute each system $\{C_0[t], C_1[t], C_2[t], C_3[t], ..., C_t[t]\}$ by an automaton $B[t]$ from K, which by the definition of function P and P-conjunctive parallel composition, works exactly as this system. Then by construction of the system H, $B = \{B[t]; t = 0, 1, 2, 3, ...\}$ is a basic evolutionary K-machine B equivalent to H. Theorem is proved. □

Corollary 3.1. *If a class K is closed with respect to P-conjunctive parallel composition, then classes GEAK and BEAK are functionally equivalent.*

The class T of all Turing machines is closed with respect to P-conjunctive parallel composition (Burgin, 2010). Thus, Theorem 3.2 implies the following result.

Corollary 3.2. *Classes GEAT of all general evolutionary Turing machines and BEAT of all basic evolutionary Turing machines are functionally equivalent.*

The class IT of all inductive Turing machines is closed with respect to P-conjunctive parallel composition (Burgin, 2010). Thus, Theorem 3.2 implies the following result.

Corollary 3.3. *Classes GEAIT of all general evolutionary inductive Turing machines and BEAIT of all basic evolutionary inductive Turing machines are functionally equivalent.*

Corollary 3.4. *Classes $GEAIT_n$ of all general evolutionary inductive Turing machines of order n and $BEAIT_n$ of all basic evolutionary inductive Turing machines of order n are functionally equivalent.*

The same is true for evolutionary limit Turing machines.

Corollary 3.5. *Classes GEALT of all general evolutionary limit Turing machines and BEALT of all basic evolutionary limit Turing machines are functionally equivalent.*

Definition 3.3. (Burgin, 2010)

a. Two automata A and B are linguistically equivalent if they accept (generate) the same language.
b. Two classes of automata A and B are linguistically equivalent if they accept (generate) the same class of languages.

For instance, it is proved that deterministic and nondeterministic finite automata are linguistically equivalent (cf,. for example (Hopcroft *et al.*, 2001)). It is proved that functional equivalence is stronger than linguistic equivalence (Burgin, 2010).

Because P-conjunctive parallel composition of the level automata in an evolutionary automaton allows the basic evolutionary K-machine to choose automata for data transmission, it is possible to prove the following results.

Theorem 3.3. *If a class K is closed with respect to P-conjunctive parallel composition, then for any general evolutionary K-machine, there is a linguistically equivalent basic evolutionary K-machine.*

Proof. Let us consider an arbitrary general evolutionary K-machine $E = \{A[t]; t = 0, 1, 2, 3, ...\}$. Then by Theorem 3.2, there is a basic evolutionary K-machine L that is functionally equivalent to E. As it is proved in (Burgin, 2010), functional equivalence implies linguistic equivalence. So, the K-machine L is linguistically equivalent to the K-machine E. Theorem is proved. □

Corollary 3.6. *If a class K is closed with respect to P-conjunctive parallel composition, then classes GEAK and BEAK are linguistically equivalent.*

The class T of all Turing machines is closed with respect to P-conjunctive parallel composition (Burgin, 2010). Thus, Theorem 3.3 implies the following result.

Corollary 3.7. *Classes GEAT of all general evolutionary Turing machines and BEAT of all basic evolutionary Turing machines are linguistically equivalent.*

The class IT of all inductive Turing machines is closed with respect to P-conjunctive parallel composition (Burgin, 2010). Thus, Theorem 3.3 implies the following results.

Corollary 3.8. *Classes GEAIT of all general evolutionary inductive Turing machines and BEAIT of all basic evolutionary inductive Turing machines are linguistically equivalent.*

Corollary 3.9. *Classes $GEAIT_n$ of all general evolutionary inductive Turing machines of order n and $BEAIT_n$ of all basic evolutionary inductive Turing machines of order n are linguistically equivalent.*

The same is true for evolutionary limit Turing machines.

Corollary 3.10. *Classes GEALT of all general evolutionary limit Turing machines and BEALT of all basic evolutionary limit Turing machines are linguistically equivalent.*

Obtained results allow us to solve the following problem formulated in (Burgin & Eberbach, 2010).

Problem 3.1. *Are periodic evolutionary finite automata more powerful than finite automata?*

To solve it, we need additional properties of periodic evolutionary finite automata.

Theorem 3.4. *Any general (basic) periodic evolutionary finite automaton F with the period $k > 1$ is functionally equivalent to a periodic evolutionary finite automaton E with the period* 1.

Proof. Let us consider an arbitrary basic periodic evolutionary finite automaton $E = \{A[t]; t = 0, 1, 2, 3, ...\}$. By the definition of basic periodic evolutionary automata (cf. Section 2), the sequence $\{A[t]; t = 0, 1, 2, 3, ...\}$ of finite automata $A[t]$ is either finite or periodic, i.e., there is a finite initial segment of this sequence such that the whole sequence is formed by infinite repetition of this segment. Note that finite sequences are also treated as periodic (Burgin & Eberbach, 2010). When the sequence $\{A[t]; t = 0, 1, 2, 3, ...\}$ of automata $A[t]$ from K is finite, then by Theorem 3.2, the evolutionary machine E is functionally equivalent to a finite automaton AE. By the definition of periodic evolutionary automata, AE is a periodic evolutionary finite automaton with the period 1. Thus, in this case, theorem is proved.

Now let us assume that the sequence $\{A[t]; t = 0, 1, 2, 3, ...\}$ of automata $A[t]$ is infinite. In this case, there is a finite initial segment $H = \{A[t]; t = 0, 1, 2, 3, ..., n\}$ of this sequence such that the whole sequence is formed by infinite repetition of this segment H. By the definition of bounded basic evolutionary automata (cf. Section 2), H is a basic n-level evolutionary finite automaton. Then by Theorem 3.1 from (Burgin & Eberbach, 2010), there is a finite automaton AH functionally equivalent to H. Thus, the evolutionary machine E is functionally equivalent to the basic periodic evolutionary finite automaton $B = \{B[t]; t = 0, 1, 2, 3, ...\}$ in which all automata $B[t] = AH$ for all $t = 0, 1, 2, 3, ...$ Thus, B is a basic periodic evolutionary finite automaton with the period 1. This concludes the proof for basic periodic evolutionary finite automata.

Now let us consider an arbitrary general periodic evolutionary finite automaton $E = \{A[t]; t = 0, 1, 2, 3, ...\}$. By the definition of general periodic evolutionary automata (cf. Section 2), the sequence $\{A[t]; t = 0, 1, 2, 3, ...\}$ of finite automata $A[t]$ is either finite or periodic, i.e., there is a finite initial segment of this sequence such that the whole sequence is formed by infinite repetition of this segment.

At first, we show that when the sequence $\{A[t]; t = 0, 1, 2, 3, ...\}$ of automata $A[t]$ from K is finite, i.e., $E = \{A[t]; t = 0, 1, 2, 3, ..., n\}$, then the evolutionary machine E is functionally equivalent to a finite automaton AE . It is possible to assume that the automata $A[t]$ use transmission

by the output when the automaton $A[t]$ uses two more symbols u_{up} and u_{dw} in its output alphabet, giving one of these symbols in its output in addition to the regular output $X[t+1]$, i.e., the output has the form (w, u_{up}) or (w, u_{dw}). If the second part of the output is u_{up} , then $A[t+1]$ sends the output generation $X[t+1]$ to $A[t+1]$. If the second part of the output is u_{dw}, then $A[t+1]$ sends the output generation $X[t+1]$ to $A[t-1]$.

We change all automata $A[t]$ to the automata $C[t]$ in the following way. If $\{q_0, q_1, q_2, ..., q_k\}$ is the set of all states of the automaton $A[t]$, then we take $\{q_{t,0}, q_{t,1}, q_{t,2}, ..., q_{t,k}\}$ as the set of all states of the automaton $C[t]$ $(t = 0, 1, 2, ..., n)$ and in the transition rules of $C[t]$, we change each q_l to $q_{t,l}$. In addition, we change the symbols u_{up} and u_{dw} to the symbols $u_{t,up}$ and $u_{t,dw}$ in the alphabet and in the transition rules of $C[t]$.

By construction, the new system $AE = \{C[t]; t = 0, 1, 2, 3, ..., n\}$ is a finite automaton functionally equivalent to the general periodic evolutionary finite automaton $E = \{A[t]; t = 0, 1, 2, 3, ..., n\}$. Then by the definition of periodic evolutionary automata (cf. Section 2), the automaton AE is a general periodic evolutionary finite automaton with the period 1. Thus, in the finite case, theorem is proved.

Now let us assume that the sequence $\{A[t]; t = 0, 1, 2, 3, ...\}$ of automata $A[t]$ is infinite. In this case, there is a finite initial segment $H = \{A[t]; t = 0, 1, 2, 3, ..., n\}$ of this sequence such that the whole sequence is formed by infinite repetition of this segment H. By the definition of bounded general evolutionary automata (cf. Section 2), H is a general n-level evolutionary finite automaton. Then as we have already proved, there is a finite automaton AH functionally equivalent to H. Thus, the evolutionary machine E is functionally equivalent to the general periodic evolutionary finite automaton $B = \{B[t]; t = 0, 1, 2, 3, ...\}$ in which all automata $B[t] = AH$ for all $t = 0, 1, 2, 3, ...$ Thus, B is a general periodic evolutionary finite automaton with the period 1. This concludes the proof for general periodic evolutionary finite automata. Theorem is proved. □

Functional equivalence implies linguistic equivalence (Burgin, 2010). Thus, Theorem 3.4 implies the following result.

Corollary 3.11. *Any general (basic) periodic evolutionary finite automaton F with the period $k > 1$ is linguistically equivalent to a periodic evolutionary finite automaton E with the period* 1.

As a periodic evolutionary finite automaton F with the period 1 consists of multiple copies of the same finite automaton, we have the following results.

Theorem 3.5. *Any basic periodic evolutionary finite automaton F is linguistically equivalent to a finite automaton.*

Proof. By Theorem 3.4, any basic periodic evolutionary finite automaton F with the period $k > 1$ is functionally equivalent to a basic periodic evolutionary finite automaton E with the period 1. It means that all levels in the evolutionary finite automaton E are copies of the same finite automaton. As a finite automaton accepts (computes) a regular language (Hopcroft *et al.*, 2001), the language of the evolutionary finite automaton E is also regular. As the evolutionary finite automaton F is linguistically equivalent to the automaton E, the language L of the evolutionary finite automaton F is also regular. Then there is a finite automaton D that accepts (computes) L (Hopcroft *et al.*, 2001). Thus, the evolutionary finite automaton F is linguistically equivalent to the finite automaton D. Theorem is proved. □

Corollary 3.12. *Basic periodic evolutionary finite automata have the same accepting power as finite automata.*

Theorem 3.6. *Any general periodic evolutionary finite automaton E is equivalent to a one-dimensional cellular automaton.*

Proof. By Theorem 3.4, any general periodic evolutionary finite automaton G with the period $k > 1$ is functionally equivalent to a general periodic evolutionary finite automaton E with the period 1. By definition, E is a sequence of copies of the same finite automaton, which each of them is connected with two its neighbors, and this is exactly a one-dimensional cellular automaton (Trahtenbrot, 1974).

At the same time, taking a finite automaton A with a feedback that connects the automaton output with the automaton input, we see that A can simulate a periodic evolutionary finite automaton E with the period 1 because in E all level automata are copies of the same finite automaton. □

In the theory of cellular automata, it is proved that for any Turing machine T, there is a cellular automaton functionally equivalent to T (Trahtenbrot, 1974). Thus, Theorem 3.6 implies the following result.

Corollary 3.13. *General periodic evolutionary finite automata have the same accepting power as Turing machines.*

Consequently, we have the following result.

Corollary 3.14. *General periodic evolutionary finite automata have more accepting power than basic periodic evolutionary finite automata and than finite automata.*

Note that we cannot apply Theorem 3.2 to periodic evolutionary finite automata because the general evolutionary machine constructed in the proof of this theorem is not periodic.

These results also allow us to solve Problem 4 from (Burgin & Eberbach, 2010).

Problem 3.2. *What class of languages is generated/accepted by periodic evolutionary finite automata?*

Namely, we have the following results.

Corollary 3.15. *The class of languages generated/accepted by basic periodic evolutionary finite automata coincides with regular languages.*

Corollary 3.16. *The class of languages generated/accepted by general periodic evolutionary finite automata coincides with recursively enumerable languages.*

Note that for unrestricted evolutionary finite automata results of Theorems 3.5, 3.6 and their corollaries are not true. Namely, we have the following result.

Theorem 3.7. *The class GEAFA of general unrestricted evolutionary finite automata and the class BEAFA of basic unrestricted evolutionary finite automata have the same accepting power.*

Proof. Indeed, as it is demonstrated in (Eberbach & Burgin, 2007), basic unrestricted evolutionary finite automata can accept any formal language. In particular, they accept any language that general unrestricted evolutionary finite automata accept. As general unrestricted evolutionary finite automata are more general than basic unrestricted evolutionary finite automata, the class of the languages accepted by the former automata is, at least, as big as the class of the languages accepted by the latter automata. Thus, these classes coincide, which means that the class of all general unrestricted evolutionary finite automata and the class of all basic unrestricted evolutionary finite automata have the same accepting power. □

The results from this paper show that in some cases, general evolutionary machines are more powerful than basic evolutionary machines, e.g., for all periodic evolutionary finite automata, while in other cases, it is not true, e.g., for all evolutionary finite automata, general and basic evolutionary finite automata have the same computing power. There are similar results in the theory of classical automata and algorithms. For instance, deterministic and nondeterministic finite automata have the same accepting power. Deterministic and nondeterministic Turing machines have the same accepting power. However, nondeterministic pushdown automata have more accepting power than deterministic pushdown automata.

4. Conclusion

We started our paper with a description of Turings unorganized machines (u-machines) that were supposed to work under the control of some kind of genetic algorithms (note that Turing never formally defined a genetic algorithm or evolutionary computation). This was our inspiration. However, our evolutionary machines are closely related to conventional Turing machines, as well as to the subsequent definitions of genetic algorithms from 1960-80s. This means that level automata of evolutionary machines are finite automata, pushdown automata or Turing machines rather than more primitive NAND logic gates of u-machines. We have introduced several classes of evolutionary machines, such as bounded, periodic and recursively generated evolutionary machines, and studied relations between these classes, giving an interpretation of how modern u-machines could be formalized and how plentiful their computations and types are. Of course, we will never know whether Turing would accept our definitions of evolutionary automata and formalization of evolutionary computation.

In this paper, we introduced two fundamental classes of evolutionary machines/automata: general evolutionary machines and basic evolutionary machines, exploring relations between these classes. Problems of generation of evolutionary machines/automata by automata from a given class are also studied. Examples of such evolutionary machines are evolutionary Turing machines generated by Turing machines and evolutionary inductive Turing machines generated by inductive Turing machines.

There are open problems important for the development of EC foundations.

Problem 4.1. *Can an inductive Turing machine of the first order simulate an arbitrary periodic evolutionary inductive Turing machine of the first order?*

Problem 4.2. *Are there necessary and sufficient conditions for general evolutionary machines to be more powerful than basic evolutionary machines?*

In (Burgin, 2001), topological computations are introduced and studied. This brings us to the following problem.

Problem 4.3. *Study topological computations for evolutionary machines.*

As we can see from results of this paper, in some cases general evolutionary machines are more powerful than basic evolutionary machines, e.g., for all evolutionary finite automata, while in other cases, it is not true, e.g., for all periodic evolutionary machines.

Note that the approach presented in this paper has an enormous space to grow. First of all, similar to natural evolution, our evolutionary automata/machines are not static, i.e., we cover the case of evolution of evolution (currently explored in a very limited way in evolution strategies by changing the σ parameter in mutation). Secondly, our evolutionary finite automata cover already both evolutionary algorithms (i.e., genetic algorithms, evolutionary programming, evolution strategies and genetic programming)and swarm intelligence algorithms, being simple iterative algorithms of the class or regular languages/finite automata. In the evolutionary automata approach, there is a room to grow to invent new types of evolutionary and swarm intelligence algorithms of the class of evolutionary pushdown automata, evolutionary Turing machines or evolutionary inductive Turing machines.

References

Burgin, M. (1983). Multiple computations and Kolmogorov complexity for such processes. *Notices of the Academy of Sciences of the USSR* **27**(2), 793–797.

Burgin, M. (2001). Topological algorithms. In: *Proceedings of the ISCA* 16^{th} *International Conference "Computers and their Applications", ISCA, Seattle, Washington*. pp. 61–64.

Burgin, M. (2005). *Super-Recursive Algorithms*. Springer-Verlag.

Burgin, M. (2010). *Measuring Power of Algorithms, Computer Programs, and Information Automata*. Nova Science Publishers, New York.

Burgin, M. and E. Eberbach (2008). Cooperative combinatorial optimization: Evolutionary computation case study. *BioSystems* **91**(1), 34–50.

Burgin, M. and E. Eberbach (2009*a*). *On Foundations of Evolutionary Computation: An Evolutionary Automata Approach*. Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies (Hongwei Mo, Ed.), IGI Global, Hershey, Pennsylvania. pp. 342-360.

Burgin, M. and E. Eberbach (2009*b*). Universality for Turing machines, inductive Turing machines and evolutionary algorithms. *Fundamenta Informaticae* **91**(1), 53–77.

Burgin, M. and E. Eberbach (2010). Bounded and periodic evolutionary machines. In: *Proc. 2010 Congress on Evolutionary Computation (CEC2010), Barcelona, Spain*. pp. 1379–1386.

Burgin, M. and E. Eberbach (2012). Evolutionary automata: Expressiveness and convergence of evolutionary computation. *Computer Journal* **55**(9), 1023–1029.

Eberbach, E. (2005). Toward a theory of evolutionary computation. *BioSystems* **82**(1), 1–19.

Eberbach, E. and M. Burgin (2007). Evolution of evolution: Self-constructing evolutionary Turing machine case study. In: *Proc. 2007 Congr. on Evolutionary Computation (CEC2007), Singapore*. pp. 4599–4604.

Eberbach, E., D. Goldin and P. Wegner (2004). *Turing's Ideas and Models of Computation*. Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag. pp. 159-194.

He, J. and X. Yao (2004). A study of drift analysis for estimating computation time of evolutionary algorithms. *Nat. Comput.* **3**, 21–25.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, Ann Arbor, MIT Press, 2nd ed.

Hopcroft, J. E., R. Motwani and J. D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley,Boston/San Francisco/New York.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press. Cambridge, MA, USA.

Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press. Cambridge, MA, USA.

Koza, John R., David Andre, Forrest H. Bennett and Martin A. Keane (1999). *Genetic Programming III: Darwinian Invention & Problem Solving*. 1st ed.. Morgan Kaufmann Publishers Inc.. San Francisco, CA, USA.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. Third edition.

Michalewicz, Z. and D.B. Fogel (2004). *How to Solve It: Modern Heuristics*. Springer-Verlag. 2nd ed.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Networks: Special Issue on EC* **5**(1), 96–101.

Trahtenbrot, B.A. (1974). *Algorithms and Computing Automata*. Moscow, Sovetskoye Radio. in Russian.

Turing, A. (1992). *Intelligent Machinery, in Collected Works of A.M. Turing: Mechanical Intelligence*. Elsevier Science.

Wolpert, D.H. and W.G. Macready (1997). No free lunch theorem for optimization. *IEEE Trans. Evol. Comput.* **1**(1), 67–82.